Problema fracionário da mochila

Problema: Dados (w, v, n, W), encontrar uma mochila ótima.

Exemplo: $W = 50$, $n = 4$

	1	2	3	4	
w	40	30	20	10	
v	840	600	400	100	
x	1	0	0	0	valor $= 840$
x	1	0	0	1	valor $= 940$
x	0	1	1	0	valor $= 1000$
x	1	1/3	0	0	valor $= 1040$

Subestrutura ótima

Suponha que $x[1\,..\,n]$ é mochila ótima
para o problema (w, v, n, W).

Se $x[n] = \delta$

então $x[1\,..\,n-1]$ é mochila ótima para

$$(w, v, n-1, W - \delta\, w[n])$$

NOTA. Não há nada de especial acerca do índice n.
Uma afirmação semelhante vale para qualquer índice i.

Então podemos fazer uma algoritmo de PD como no caso anterior.
Mas não dá para fazer algo mais simples, melhor?

Escolha gulosa

Suponha $w[i] \neq 0$ para todo i.

Escolha gulosa

Suponha $w[i] \neq 0$ para todo i.

Se $v[n]/w[n] \geq v[i]/w[i]$ para todo i

então EXISTE uma mochila ótima $x[1\,..\,n]$ tal que

$$x[n] = \min \left\{ 1, \frac{W}{w[n]} \right\}$$

Algoritmo guloso

Esta propriedade da escolha gulosa sugere
um algoritmo que atribui os valores de $x[1\,..\,n]$ supondo que
os dados estejam em ordem decrescente de "valor específico":

$$\frac{v[1]}{w[1]} \leq \frac{v[2]}{w[2]} \leq \ldots \leq \frac{v[n]}{w[n]}$$

Algoritmo guloso

Esta propriedade da escolha gulosa sugere
um algoritmo que atribui os valores de $x[1\,..\,n]$ supondo que
os dados estejam em ordem decrescente de "valor específico":

$$\frac{v[1]}{w[1]} \leq \frac{v[2]}{w[2]} \leq \ldots \leq \frac{v[n]}{w[n]}$$

É nessa ordem "mágica" que está
o segredo do funcionamento do algoritmo.

Algoritmo guloso

Devolve uma mochila ótima para (w, v, n, W).

```
MOCHILA-FRACIONÁRIA (w, v, n, W)
 0  ordene w e v de tal forma que
        v[1]/w[1] ≤ v[2]/w[2] ≤ ··· ≤ v[n]/w[n]
 1  para i ← n decrescendo até 1 faça
 2      se w[i] ≤ W
 3          então x[i] ← 1
 4                W ← W − w[i]
 5          senão x[i] ← W/w[i]
 6                W ← 0
 7  devolva x
```

Algoritmo guloso

Devolve uma mochila ótima para (w, v, n, W).

```
MOCHILA-FRACIONÁRIA (w, v, n, W)
 0  ordene w e v de tal forma que
        v[1]/w[1] ≤ v[2]/w[2] ≤ ··· ≤ v[n]/w[n]
 1  para i ← n decrescendo até 1 faça
 2      se w[i] ≤ W
 3          então x[i] ← 1
 4                W ← W − w[i]
 5          senão x[i] ← W/w[i]
 6                W ← 0
 7  devolva x
```

Consumo de tempo da linha 0 é $\Theta(n \lg n)$.

Consumo de tempo das linhas 1–7 é $\Theta(n)$.

Invariante

Seja W_0 o valor original de W.

No início de cada execução da linha 1 vale que

Invariante

Seja W_0 o valor original de W.

No início de cada execução da linha 1 vale que

(i0) $x' = x[i{+}1\,..\,n]$ é mochila ótima para

$$(w', v', n', W_0)$$

onde

$$w' = w[i{+}1\,..\,n]$$
$$v' = v[i{+}1\,..\,n]$$
$$n' = n - i$$

Invariante

Seja W_0 o valor original de W.

No início de cada execução da linha 1 vale que

(i0) $x' = x[i+1\,..\,n]$ é mochila ótima para

$$(w', v', n', W_0)$$

onde

$$w' = w[i+1\,..\,n]$$
$$v' = v[i+1\,..\,n]$$
$$n' = n - i$$

Na última iteração $i = 0$ e
portanto $x[1\,..\,n]$ é mochila ótima para (w, v, n, W_0).

Conclusão

O consumo de tempo do algoritmo
MOCHILA-FRACIONÁRIA é $\Theta(n \lg n)$.

Escolha gulosa

Precisamos mostrar que

se $x[1\,..\,n]$ é uma mochila ótima,

então podemos supor que

$$x[n] = \alpha := \min \left\{ 1, \frac{W}{w[n]} \right\}.$$

Escolha gulosa

Precisamos mostrar que

se $x[1\,..\,n]$ é uma mochila ótima,

então podemos supor que

$$x[n] = \alpha := \min \left\{ 1, \frac{W}{w[n]} \right\}.$$

Depois de mostrar isto, indução faz o resto do serviço.

Escolha gulosa

Precisamos mostrar que
se $x[1\,..\,n]$ é uma mochila ótima,
então podemos supor que

$$x[n] = \alpha := \min\left\{1, \frac{W}{w[n]}\right\}.$$

Depois de mostrar isto, indução faz o resto do serviço.

Técnica:
transformar uma solução ótima em uma solução ótima 'gulosa'.

Escolha gulosa

Precisamos mostrar que
se $x[1\,..\,n]$ é uma mochila ótima,
então podemos supor que

$$x[n] = \alpha := \min\left\{1, \frac{W}{w[n]}\right\}.$$

Depois de mostrar isto, indução faz o resto do serviço.

Técnica:
transformar uma solução ótima em uma solução ótima 'gulosa'.

Esta transformação é semelhante ao processo de pivotação
do algoritmo SIMPLEX para programação linear.

Escolha gulosa

Seja $x[1\,..\,n]$ uma mochila ótima para (w, v, n, W)
tal que $x[n]$ é máximo. Se $x[n] = \alpha$, não há o que mostrar.

Escolha gulosa

Seja $x[1\,..\,n]$ uma mochila ótima para (w, v, n, W)
tal que $x[n]$ é máximo. Se $x[n] = \alpha$, não há o que mostrar.

Suponha $x[n] < \alpha$.

Escolha gulosa

Seja $x[1\,..\,n]$ uma mochila ótima para (w, v, n, W)
tal que $x[n]$ é máximo. Se $x[n] = \alpha$, não há o que mostrar.

Suponha $x[n] < \alpha$.

Podemos supor que $x \cdot w = W$. (Podemos mesmo?)

Escolha gulosa

Seja $x[1\,..\,n]$ uma mochila ótima para (w, v, n, W)
tal que $x[n]$ é máximo. Se $x[n] = \alpha$, não há o que mostrar.

Suponha $x[n] < \alpha$.

Podemos supor que $x \cdot w = W$. (Podemos mesmo?)

Seja i em $[1\,..\,n{-}1]$ tal que $x[i] > 0$.
(Quem garante que existe um tal i?).

Escolha gulosa

Seja $x[1\,..\,n]$ uma mochila ótima para (w, v, n, W)
tal que $x[n]$ é máximo. Se $x[n] = \alpha$, não há o que mostrar.

Suponha $x[n] < \alpha$.

Podemos supor que $x \cdot w = W$. (Podemos mesmo?)

Seja i em $[1\,..\,n{-}1]$ tal que $x[i] > 0$.
(Quem garante que existe um tal i?).

Seja

$$\delta := \min \left\{ x[i], (\alpha - x[n]) \frac{w[n]}{w[i]} \right\}$$

e

$$\beta := \delta \, \frac{w[i]}{w[n]} \, .$$

Escolha gulosa

Seja $x[1\,..\,n]$ uma mochila ótima para (w, v, n, W)
tal que $x[n]$ é máximo. Se $x[n] = \alpha$, não há o que mostrar.

Suponha $x[n] < \alpha$.

Podemos supor que $x \cdot w = W$. (Podemos mesmo?)

Seja i em $[1\,..\,n{-}1]$ tal que $x[i] > 0$.
(Quem garante que existe um tal i?).

Seja

$$\delta := \min \left\{ x[i], (\alpha - x[n]) \frac{w[n]}{w[i]} \right\}$$

e

$$\beta := \delta \, \frac{w[i]}{w[n]} \, .$$

Note que $\delta > 0$ e $\beta > 0$.

Mais escolha gulosa

Seja $x'[1\,..\,n]$ tal que

$$x'[j] := \begin{cases} x[j] & \text{se } j \neq i \text{ e } j \neq n, \\ x[i] - \delta & \text{se } j = i, \\ x[n] + \beta & \text{se } j = n. \end{cases}$$

Mais escolha gulosa

Seja $x'[1\,..\,n]$ tal que

$$x'[j] := \begin{cases} x[j] & \text{se } j \neq i \text{ e } j \neq n, \\ x[i] - \delta & \text{se } j = i, \\ x[n] + \beta & \text{se } j = n. \end{cases}$$

Verifique que $0 \leq x[j] \leq 1$ para todo j.

Mais escolha gulosa

Seja $x'[1\,..\,n]$ tal que

$$x'[j] := \begin{cases} x[j] & \text{se } j \neq i \text{ e } j \neq n, \\ x[i] - \delta & \text{se } j = i, \\ x[n] + \beta & \text{se } j = n. \end{cases}$$

Verifique que $0 \leq x[j] \leq 1$ para todo j.

Além disso, temos que

$$\begin{aligned} x' \cdot w &= x'[1]w[1] + \cdots + x'[i]w[i] + \cdots + x'[n]w[n] \\ &= x[1]w[1] + \cdots + (x[i] - \delta)w[i] + \cdots + (x[n] + \beta)w[n] \\ &= x[1]w[1] + \cdots + (x[i] - \delta)w[i] + \cdots + \left(x[n] + \delta \frac{w[i]}{w[n]}\right) w[n] \\ &= x[1]w[1] + \cdots + x[i]w[i] - \delta w[i] + \cdots + x[n]w[n] + \delta w[i] \\ &= W\,. \end{aligned}$$

Mais escolha gulosa ainda

Temos ainda que

$$
\begin{aligned}
x' \cdot v &= x'[1]v[1] + \cdots + x'[i]v[i] + \cdots + x'[n]v[n] \\
&= x[1]v[1] + \cdots + (x[i] - \delta)v[i] + \cdots + (x[n] + \beta)v[n] \\
&= x[1]v[1] + \cdots + (x[i] - \delta)v[i] + \cdots + \left(x[n] + \delta \frac{w[i]}{w[n]}\right) v[n] \\
&= x[1]v[1] + \cdots + x[i]v[i] - \delta v[i] + \cdots + x[n]v[n] + \delta w[i] \frac{v[n]}{w[n]} \\
&= x \cdot v + \delta \left(w[i] \frac{v[n]}{w[n]} - v[i] \right) \\
&\geq x \cdot v + \delta (w[i] \frac{v[i]}{w[i]} - v[i]) \quad \text{(devido à escolha gulosa!)} \\
&= x \cdot v.
\end{aligned}
$$

Escolha gulosa: epílogo

Assim, x' é uma mochila tal que $x' \cdot v \geq x \cdot v$.

Escolha gulosa: epílogo

Assim, x' é uma mochila tal que $x' \cdot v \geq x \cdot v$.
Como x é mochila ótima, concluímos que $x' \cdot v = x \cdot v$ e que x' é uma mochila ótima que contradiz a nossa escolha de x, já que

$$x'[n] = x[n] + \beta > x[n].$$

Escolha gulosa: epílogo

Assim, x' é uma mochila tal que $x' \cdot v \geq x \cdot v$.
Como x é mochila ótima, concluímos que $x' \cdot v = x \cdot v$ e que x' é uma mochila ótima que contradiz a nossa escolha de x, já que

$$x'[n] = x[n] + \beta > x[n].$$

Conclusão

Se $v[n]/w[n] \geq v[i]/w[i]$ para todo i
e x é uma mochila ótima para (w, v, n, W)
com $x[n]$ máximo, então

$$x[n] = \min \left\{ 1, \frac{W}{w[n]} \right\}.$$

A propósito . . .

O problema fracionário da mochila é um problema de programação linear (PL): encontrar um vetor x que

$$\begin{array}{rlll} \text{maximize} & x \cdot v & & \\ \text{sob as restrições} & x \cdot w & \leq & W \\ & x[i] & \geq & 0 \quad \text{para } i = 1, \ldots, n \\ & x[i] & \leq & 1 \quad \text{para } i = 1, \ldots, n \end{array}$$

A propósito . . .

O problema fracionário da mochila é um problema de programação linear (PL): encontrar um vetor x que

$$\begin{array}{rlll} \text{maximize} & x \cdot v & & \\ \text{sob as restrições} & x \cdot w & \leq & W \\ & x[i] & \geq & 0 \quad \text{para } i = 1, \ldots, n \\ & x[i] & \leq & 1 \quad \text{para } i = 1, \ldots, n \end{array}$$

PL's podem ser resolvidos por

SIMPLEX: no pior caso consome tempo exponencial
na prática é muito rápido

ELIPSÓIDES: consome tempo polinomial na prática é lento

PONTOS-INTERIORES: consome tempo polinomial
na prática é rápido

Algoritmos gulosos (*greedy*)

CLRS 16.3

Códigos de Huffman

Motivação: compressão de textos

Códigos de Huffman

Motivação: compressão de textos

Código ASCII: todo símbolo tem um código de 8 bits.

Códigos de Huffman

Motivação: compressão de textos

Código ASCII: todo símbolo tem um código de 8 bits.

Σ: alfabeto finito

f_i: frequência de símbolo i em Σ
(coleção de números não-negativos cuja soma é 1)

Códigos de Huffman

Motivação: compressão de textos

Código ASCII: todo símbolo tem um código de 8 bits.

Σ: alfabeto finito

f_i: frequência de símbolo i em Σ
(coleção de números não-negativos cuja soma é 1)

Objetivo: atribuir um código binário para cada símbolo de modo que um texto seja convertido para um arquivo binário o mais compacto possível e seja fácil de decodificar.

Códigos de Huffman

Motivação: compressão de textos

Código ASCII: todo símbolo tem um código de 8 bits.

Σ: alfabeto finito

f_i: frequência de símbolo i em Σ
(coleção de números não-negativos cuja soma é 1)

Objetivo: atribuir um código binário para cada símbolo
de modo que um texto seja convertido para um arquivo
binário o mais compacto possível e seja fácil de decodificar.

Códigos livres de prefixo: o código de um símbolo
não é prefixo do código de nenhum outro símbolo.

Códigos de Huffman

Motivação: compressão de textos

Código ASCII: todo símbolo tem um código de 8 bits.

Σ: alfabeto finito

f_i: frequência de símbolo i em Σ
(coleção de números não-negativos cuja soma é 1)

Objetivo: atribuir um código binário para cada símbolo de modo que um texto seja convertido para um arquivo binário o mais compacto possível e seja fácil de decodificar.

Códigos livres de prefixo: o código de um símbolo não é prefixo do código de nenhum outro símbolo.

Códigos livres de prefixo são fáceis de decodificar.

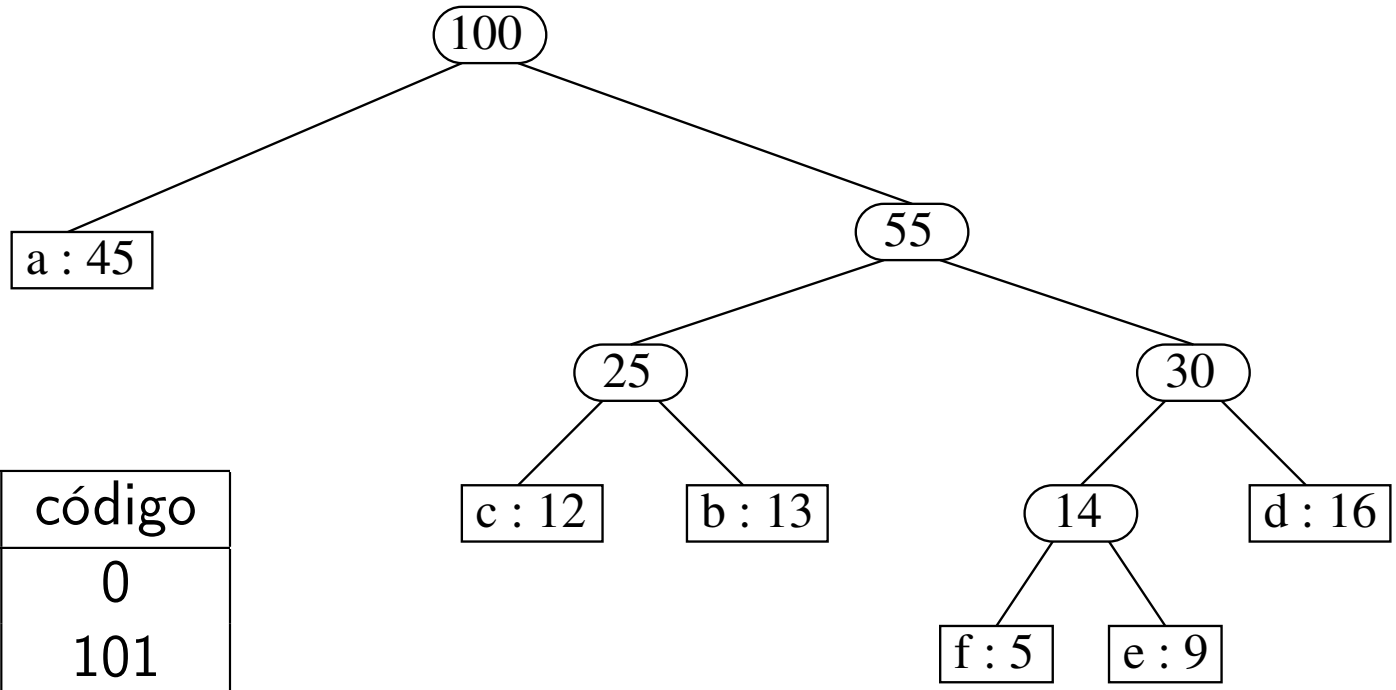
Exemplo

letra	freq	código
a	45	
b	13	
c	12	
d	16	
e	9	
f	5	

Exemplo

letra	freq	código
a	45	0
b	13	101
c	12	100
d	16	111
e	9	1101
f	5	1100

Exemplo



letra	freq	código
a	45	0
b	13	101
c	12	100
d	16	111
e	9	1101
f	5	1100

Algoritmo de Huffman

n: número de símbolos em Σ

Algoritmo de Huffman

n: número de símbolos em Σ

Guloso:

Comece com n árvores disjuntas, cada uma com um único nó, com o símbolo e sua frequência.

a : 45	d : 16	b : 13	c : 12	e : 9	f : 5

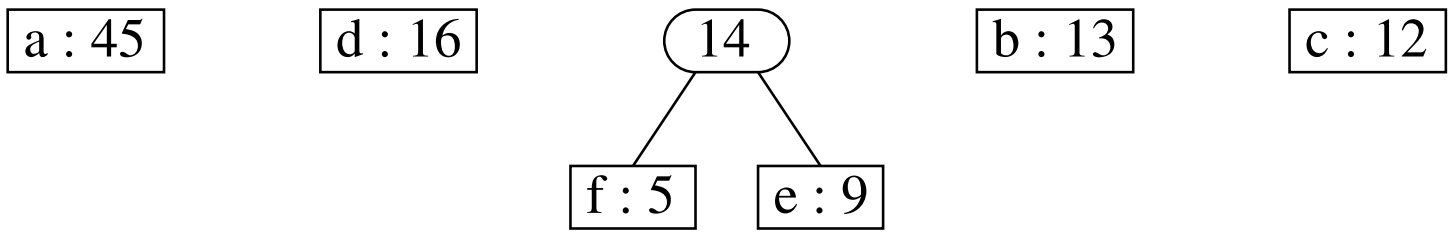
Algoritmo de Huffman

n: número de símbolos em Σ

Guloso:

Comece com n árvores disjuntas, cada uma com um único nó, com o símbolo e sua frequência.

A cada iteração, escolha as duas árvores de frequência menor e junte-as, com frequência somada.



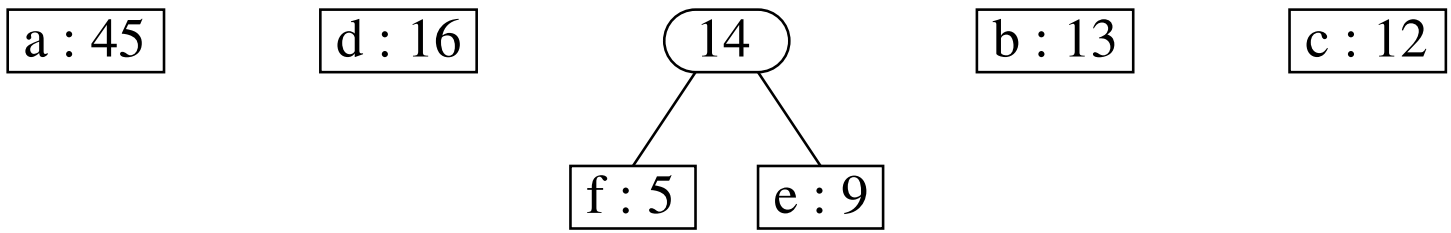
Algoritmo de Huffman

n: número de símbolos em Σ

Guloso:

Comece com n árvores disjuntas, cada uma com um único nó, com o símbolo e sua frequência.

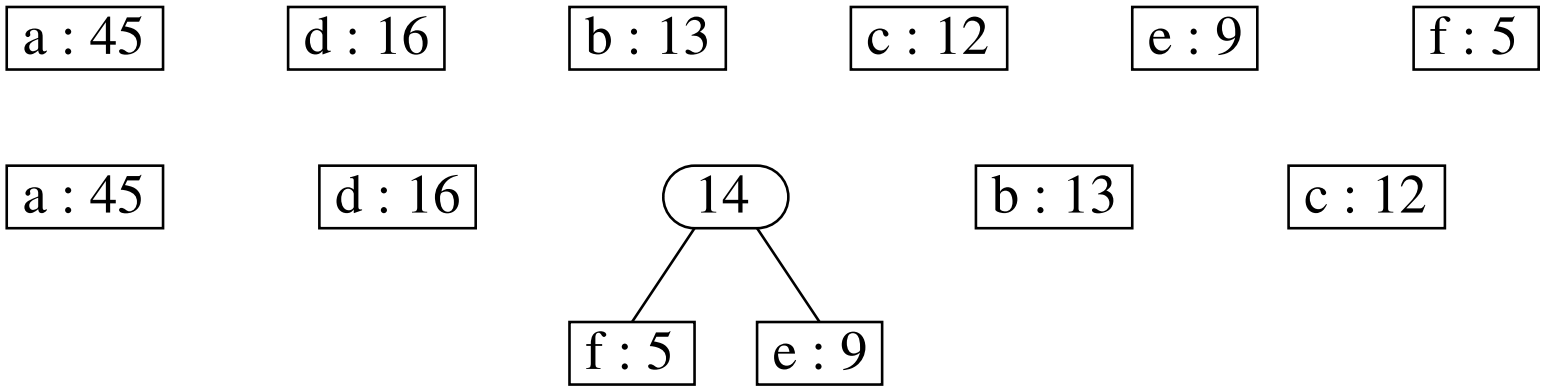
A cada iteração, escolha as duas árvores de frequência menor e junte-as, com frequência somada.



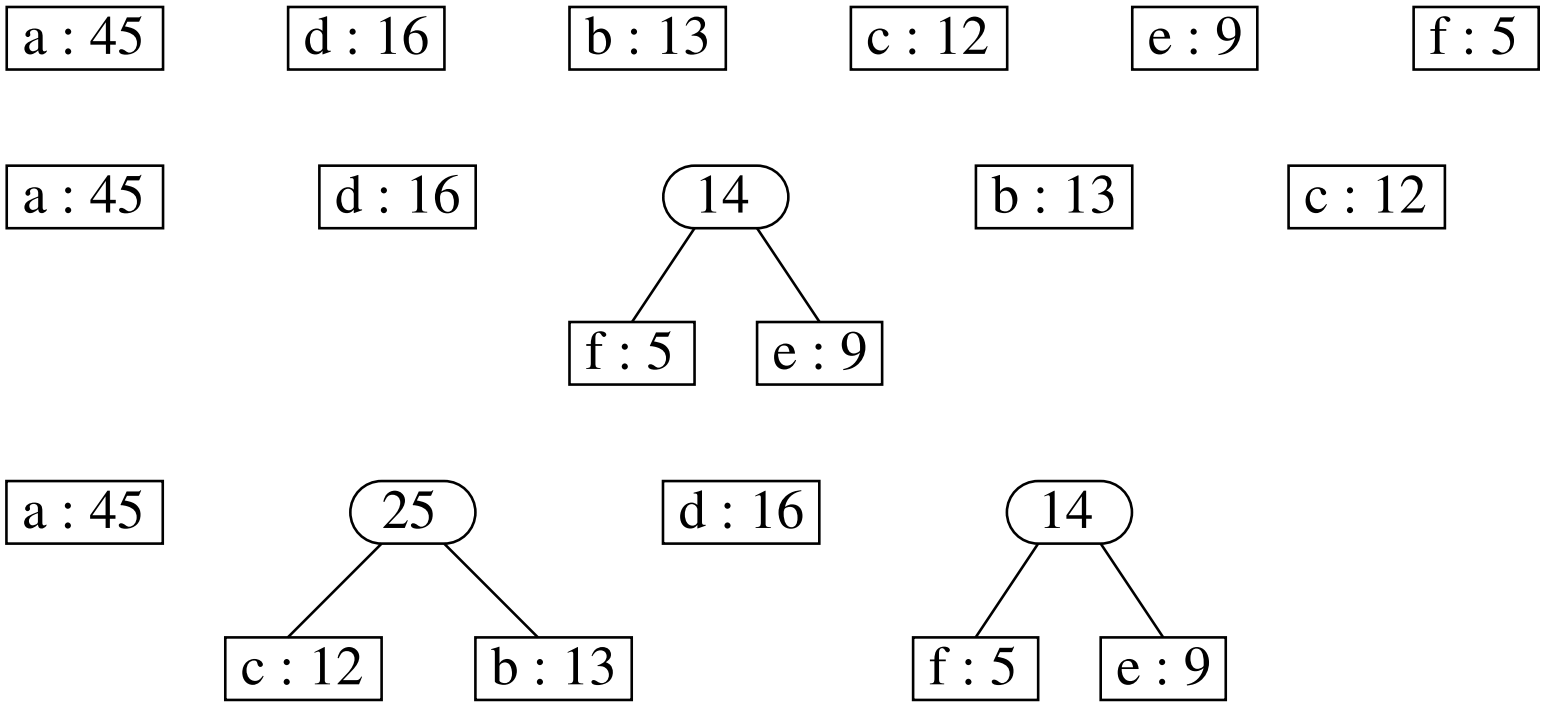
Pare quando restar uma única árvore.

Algoritmo de Huffman: exemplo

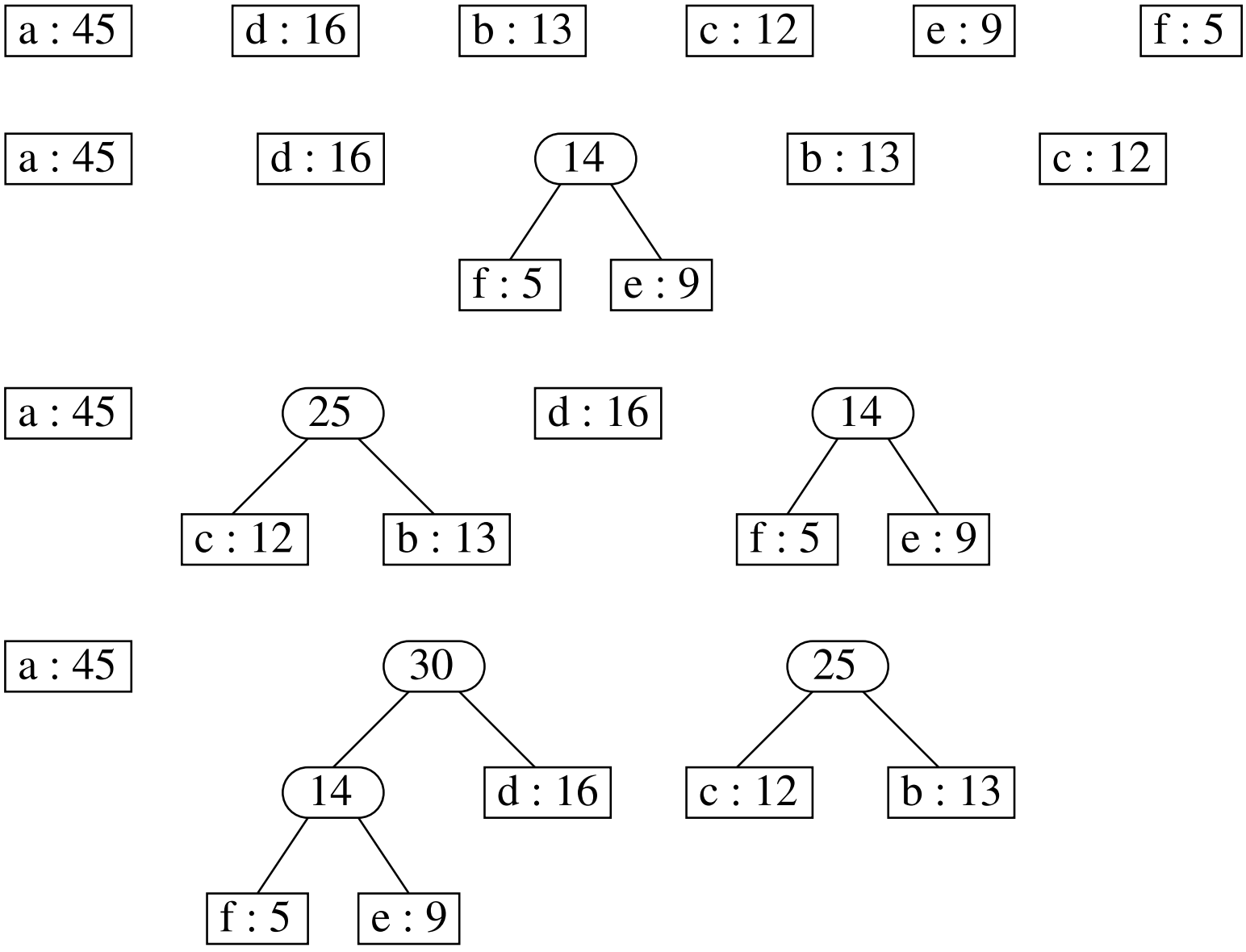
Algoritmo de Huffman: exemplo



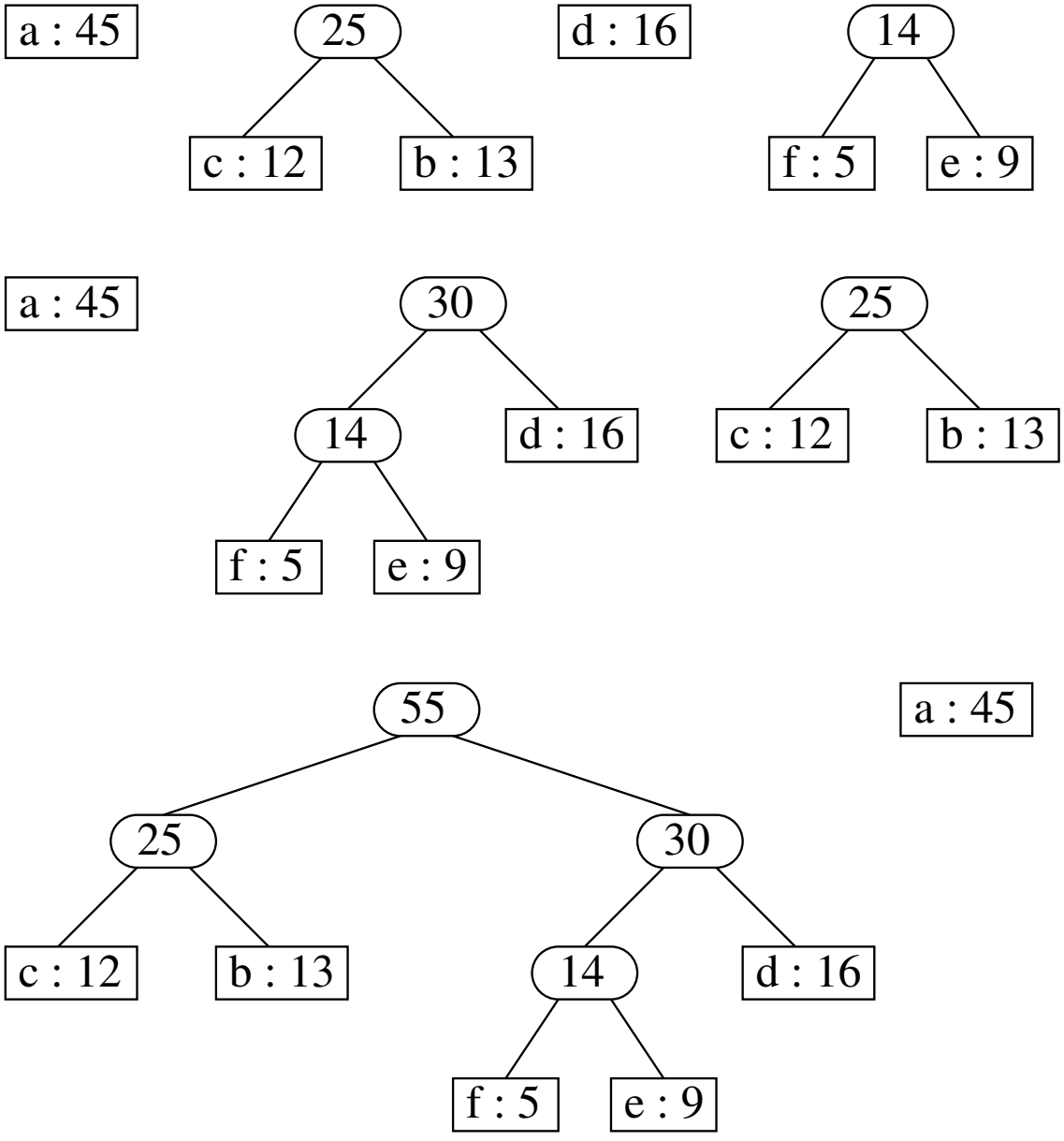
Algoritmo de Huffman: exemplo



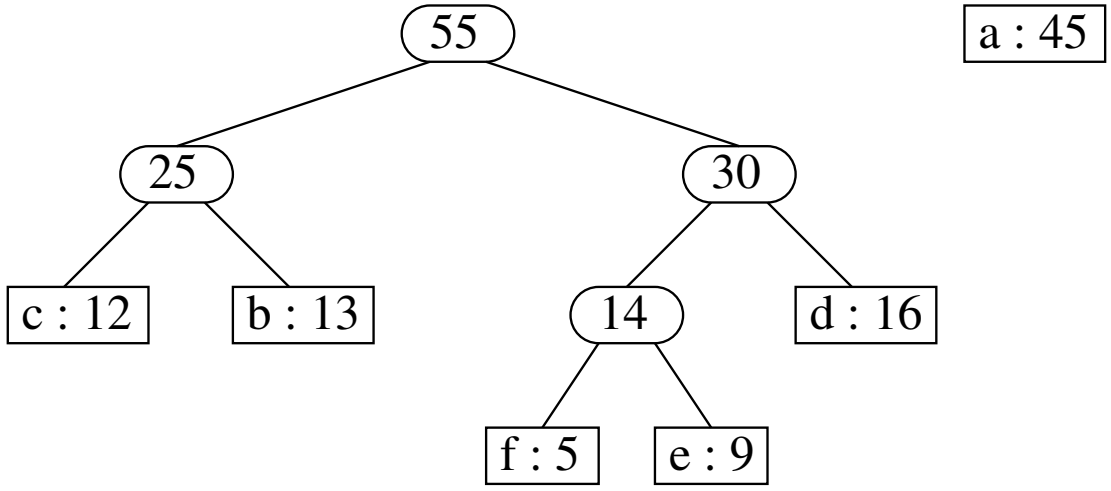
Algoritmo de Huffman: exemplo



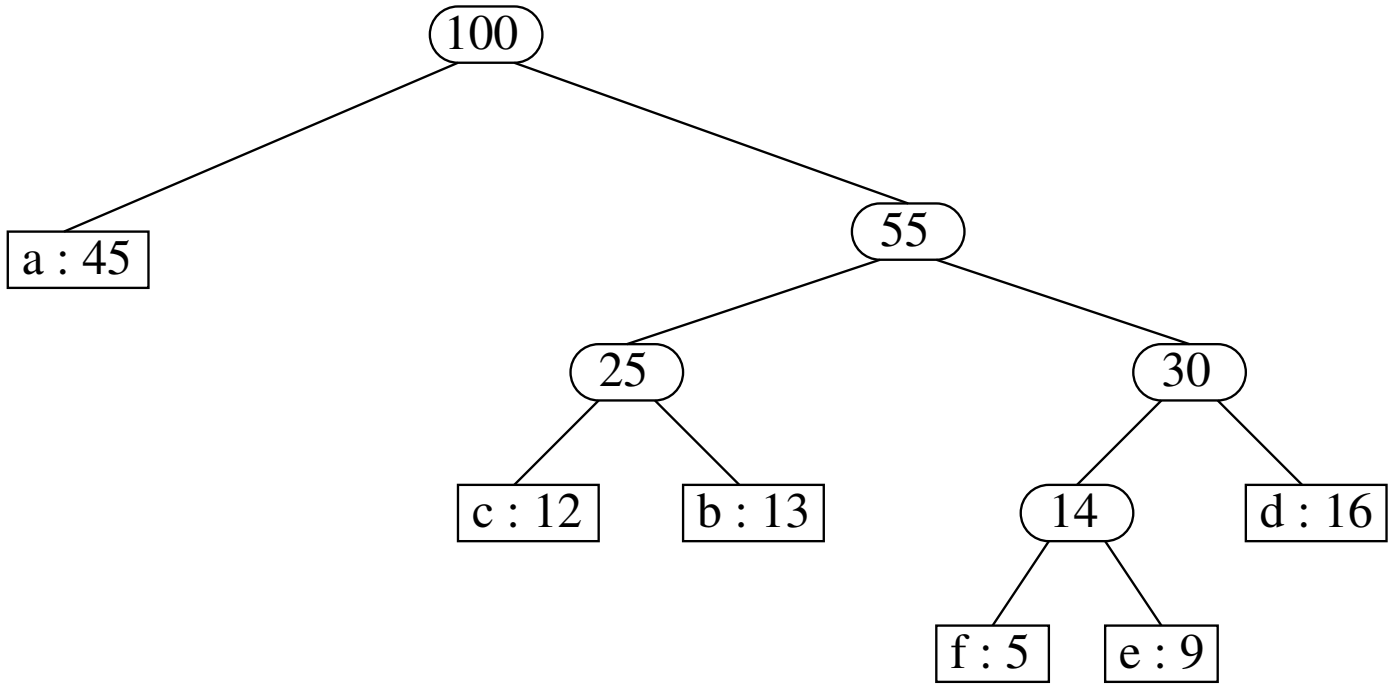
Algoritmo de Huffman: exemplo



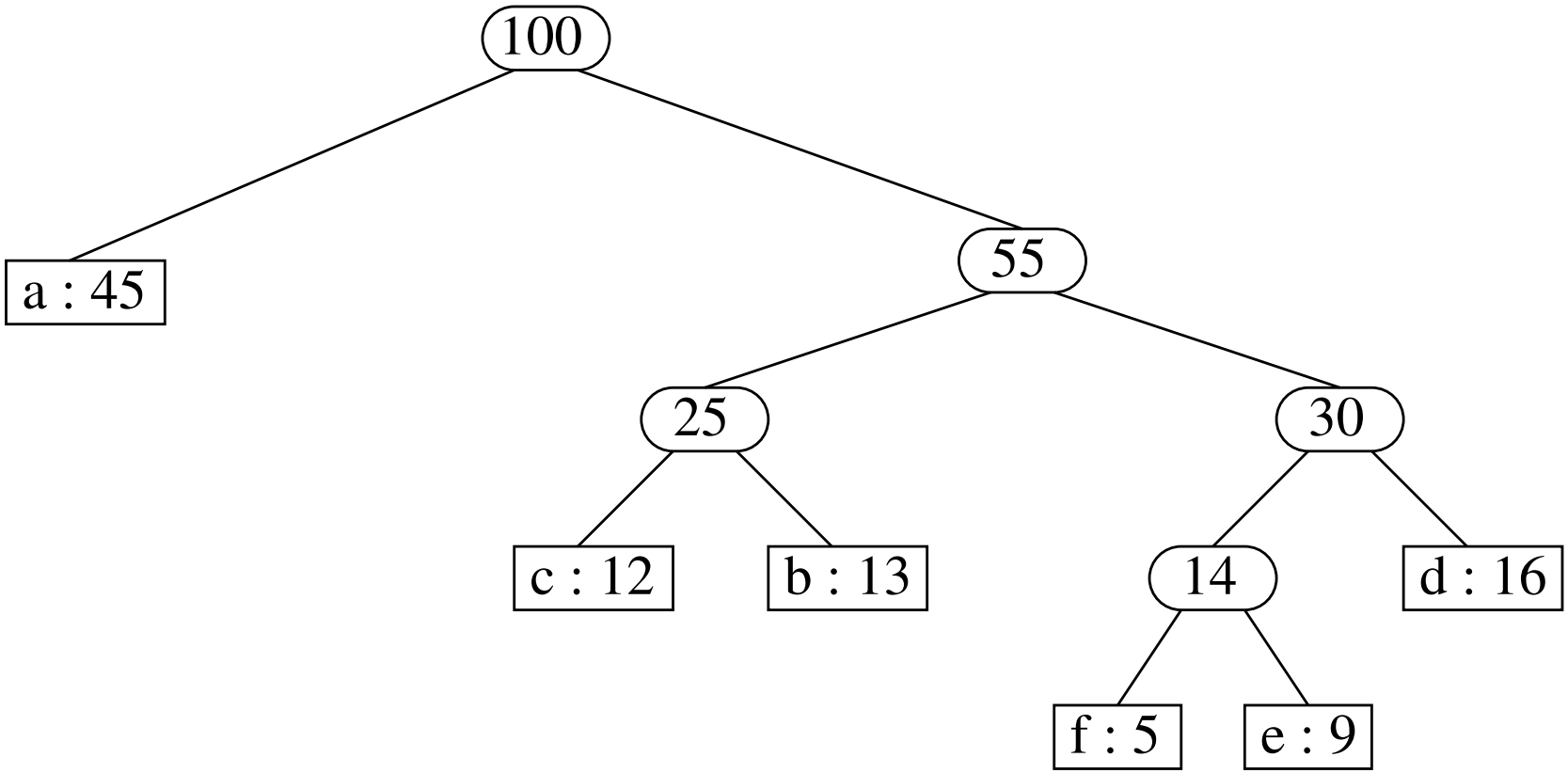
Algoritmo de Huffman: exemplo



a : 45



Árvore de Huffman



Árvore de Huffman

Como obter os códigos a partir da árvore?

Árvore de Huffman

Como obter os códigos a partir da árvore?

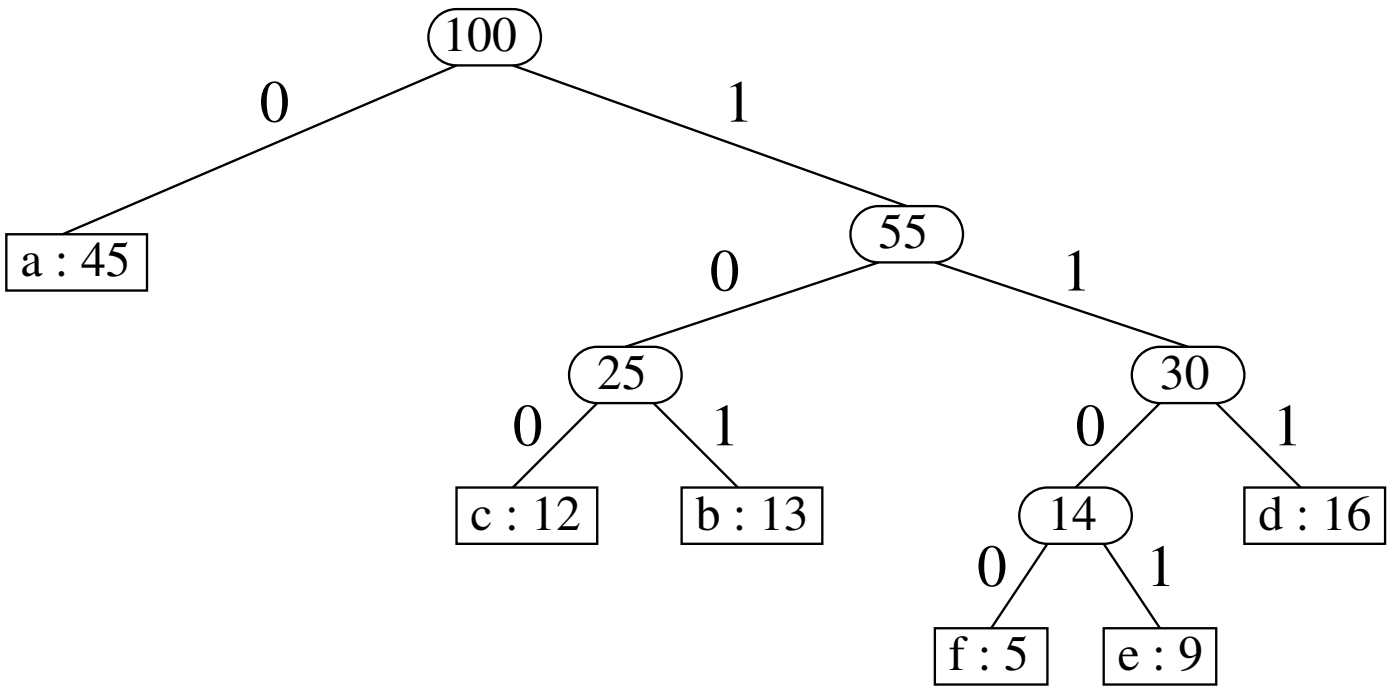
Associe a cada símbolo um número binário assim:

Árvore de Huffman

Como obter os códigos a partir da árvore?

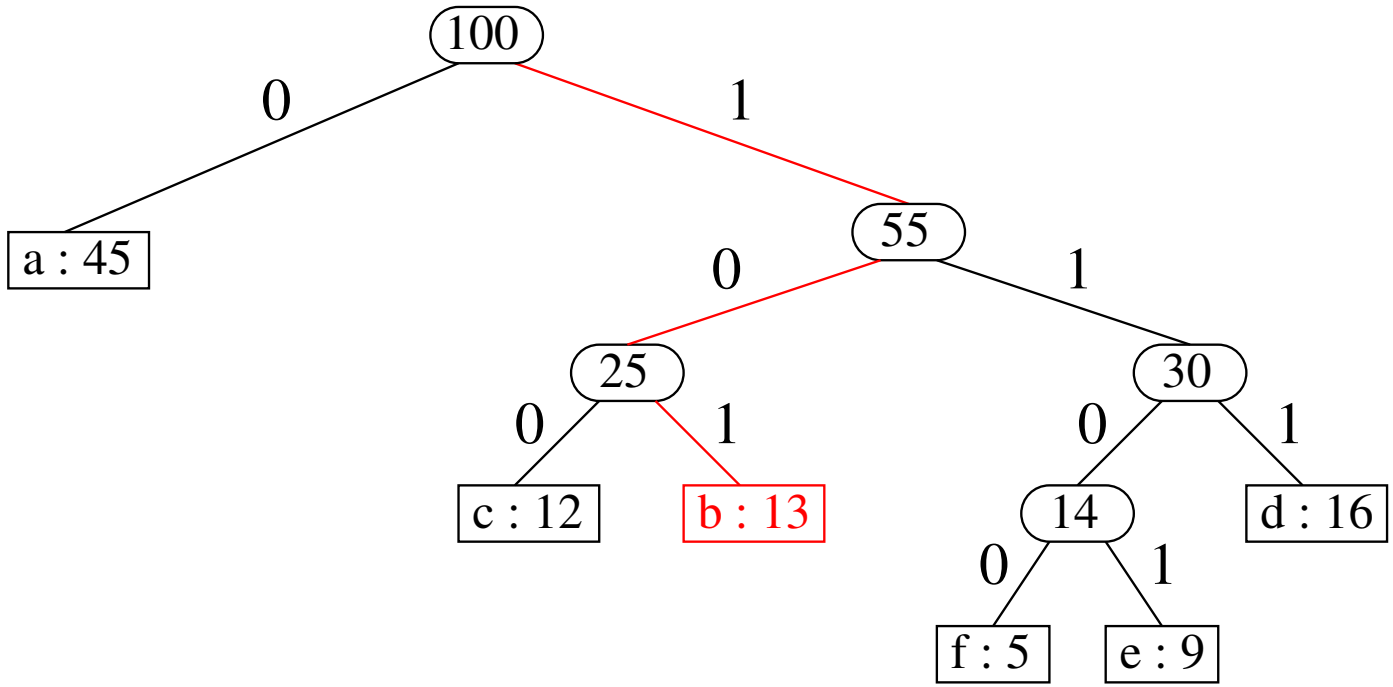
Associe a cada símbolo um número binário assim:

Rotule com 0 as arestas da árvore
que ligam um nó com seu filho esquerdo e
com 1 as arestas que ligam um nó com seu filho direito.



Árvore de Huffman

Como obter os códigos a partir da árvore?



O código correspondente a cada símbolo é a concatenação dos *bits* associados às arestas do caminho da raiz até a folha correspondente ao símbolo.

Exemplo: O código de b é 101.

Algoritmo de Huffman

n: número de símbolos em Σ

Guloso:

Comece com n árvores disjuntas, cada uma com um único nó, com o símbolo e sua frequência.

A cada iteração, escolha as duas árvores de frequência menor e junte-as, com frequência somada.

Pare quando restar uma única árvore.

Algoritmo de Huffman

n: número de símbolos em Σ

Guloso:

Comece com n árvores disjuntas, cada uma com um único nó, com o símbolo e sua frequência.

A cada iteração, escolha as duas árvores de frequência menor e junte-as, com frequência somada.

Pare quando restar uma única árvore.

Perguntas:

- Este algoritmo produz um código ótimo?
- Como implementá-lo do modo mais eficiente possível?

Algoritmo guloso

```
HUFFMAN (A, f, n)
 1  Q ← BUILD-MIN-HEAP(A, f, n)
 2  para i ← 1 até n − 1 faça
 3      x ← EXTRACT-MIN(Q)
 4      y ← EXTRACT-MIN(Q)
 5      z ← NOVA-CEL()
 6      esq[z] ← x
 7      dir[z] ← y
 8      f[z] ← f[x] + f[y]
 9      INSEREHEAP(Q, z, f[z])
10  devolva EXTRACT-MIN(Q)
```

Algoritmo guloso

```
HUFFMAN (A, f, n)
 1  Q ← BUILD-MIN-HEAP(A, f, n)
 2  para i ← 1 até n − 1 faça
 3      x ← EXTRACT-MIN(Q)
 4      y ← EXTRACT-MIN(Q)
 5      z ← NOVA-CEL()
 6      esq[z] ← x
 7      dir[z] ← y
 8      f[z] ← f[x] + f[y]
 9      INSEREHEAP(Q, z, f[z])
10  devolva EXTRACT-MIN(Q)
```

Consumo de tempo: $\mathrm{O}(n \lg n)$.

Coloração de intervalos

Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$ e um inteiro positivo k, uma k-coloração dos intervalos é uma partição deles em k coleções de intervalos disjuntos.

Coloração de intervalos

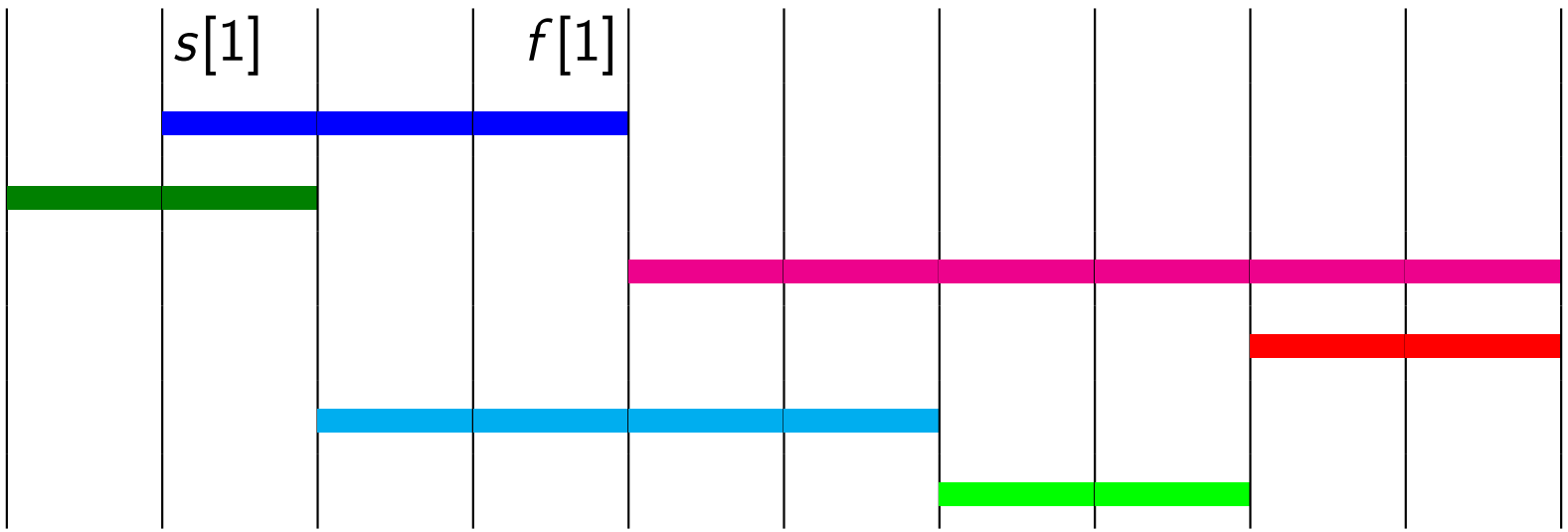
Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$ e um inteiro positivo k, uma k-coloração dos intervalos é uma partição deles em k coleções de intervalos disjuntos.

Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$, encontrar uma k-coloração dos intervalos com k o menor possível.

Coloração de intervalos

Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$ e um inteiro positivo k, uma k-coloração dos intervalos é uma partição deles em k coleções de intervalos disjuntos.

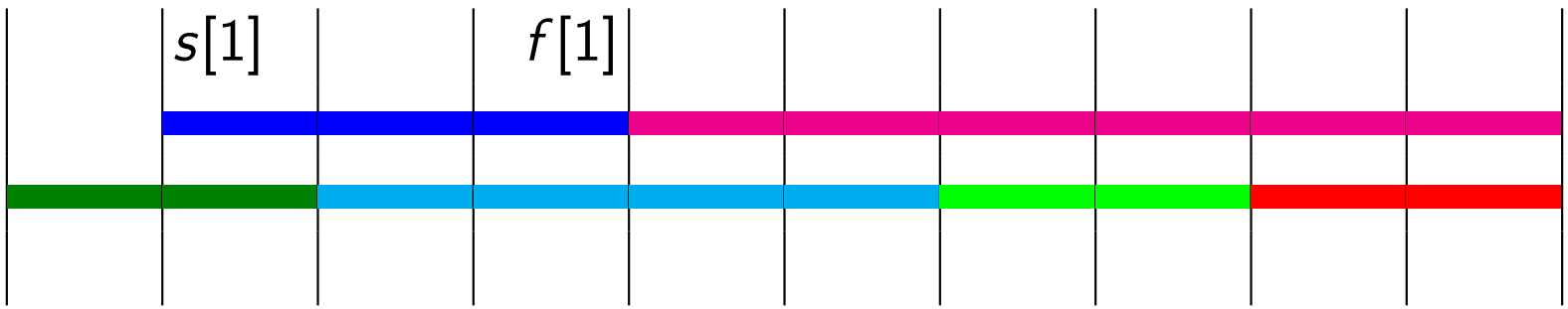
Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$, encontrar uma k-coloração dos intervalos com k o menor possível.



Coloração de intervalos

Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$ e um inteiro positivo k, uma k-coloração dos intervalos é uma partição deles em k coleções de intervalos disjuntos.

Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$, encontrar uma k-coloração dos intervalos com k o menor possível.



Solução: 2-coloração.

Coloração de intervalos

Estratégias gulosas:

- Encontre uma coleção disjunta máxima de intervalos, pinte com a próxima cor disponível e repita a idéia para os intervalos restantes.
- Ordene as atividades de maneira que $f[1] \leq f[2] \leq \cdots \leq f[n]$ e pinte uma a uma nesta ordem sempre usando a menor cor possível para aquela atividade.
- Ordene as atividades de maneira que $s[1] \leq s[2] \leq \cdots \leq s[n]$ e pinte uma a uma nesta ordem sempre usando a menor cor possível para aquela atividade.

Quais destas estratégias funcionam?

Quais não funcionam?

Algoritmo guloso

```
GULOSO (s, f, n)
 1  ORDENE(s, f, n)     ▷ s[1] ≤ s[2] ≤ ··· ≤ s[n]
 2  m ← 0
 3  ℓ[1] ← 0
 4  para k ← 1 até n faça
 5      j ← 1
 6      enquanto ℓ[j] > s[k] faça
 7          j ← j + 1
 8      se j > m
 9          então m ← j
10                  ℓ[m + 1] ← 0
11      ℓ[j] ← f[k]
12      c[k] ← j
13  devolva c
```

Exercícios

1. Mostre um exemplo para os três primeiros critérios gulosos apresentados para o primeiro problema que prove que o algoritmo obtido usando estes critérios pode produzir um conjunto A que não é máximo.
2. Considere o algoritmo do slide anterior para o segundo problema. Modifique-o para que, além de c, ele devolva um conjunto S de m intervalos e um instante t tal que $s[i] \leq t < f[i]$ para todo i em S.
3. Numa versão anterior dos slides desta aula, o algoritmo do segundo problema ordenava os intervalos pelo valor de f em vez de pelo valor de s. Isso faz diferença? Ou seja, a versão do algoritmo da página anterior que ordena os intervalos por f em vez de s também dá uma resposta correta? Prove ou dê um contra-exemplo.